

ESTRATEGIAS DE DESARROLLO DE PROYECTOS CON MICROCONTROLADORES USANDO HERRAMIENTAS *OPEN SOURCE* Y GENERADORES DE CÓDIGO

Salvador Eduardo Tropea

Unidad Técnica Instrumentación y Control, CITEI, INTI, Argentina salvador@inti.gov.ar

ABSTRACT

En este trabajo se expone como metodología de trabajo el uso de herramientas *open source* y generadores de código para el desarrollo de aplicaciones con microcontroladores PIC de Microchip.

1. INTRODUCCIÓN

Se necesitaba desarrollar un medidor de gases (CO y CH₄) domiciliario que dispusiera de un *display* alfanumérico, indicadores sonoros y luminosos y un botón de *reset* y configuración del dispositivo.

Por cuestiones de costo, volumen de producción y experiencia se decidió implementarlo utilizando un microcontrolador (MCU) PIC de Microchip. La MCU seleccionada fue la 16C716 que en ese momento era muy nuevo y carecía de un soporte adecuado en todas las herramientas disponibles.

2. MÉTODO EMPLEADO

Teniendo en cuenta, entre otras cosas, la estabilidad del sistema de desarrollo, costos y flexibilidad se decidió utilizar herramientas *open source* para el desarrollo.

Debido a la necesidad de flexibilidad se decidió utilizar generadores de código, esto se explica con mayor detalle en la sección 4.

3. HERRAMIENTA UTILIZADAS

3.1. Programador

Los programadores que disponíamos para DOS no soportaban esta MCU y el fabricante era virtualmente imposible de contactar. El que poseíamos para Windows tampoco la soportaba y estaba discontinuado. De no haber elegido la metodología *open source* deberíamos haber comprado un nuevo programador. Esto no afectaría a este proyecto pero en muchas áreas, por ejemplo la educativa, es muy pesado comprar nuevos programadores cada vez que se use un nuevo dispositivo. Es cierto que algunos programadores se pueden actualizar, sin embargo esto

implica un costo considerable, especialmente en mercados locales.

Usamos un programa llamado *prog84*[1] y un hardware que ya teníamos. Prog84 tiene la ventaja de ser completamente configurable de manera tal que no fue difícil hacer que aceptara nuestro hardware. El 16C716 no estaba soportado, pero se solucionó haciendo pequeñas modificaciones en el programa, también fue necesario introducir cambios para soportar señales especiales que poseía nuestro hardware. Prog84 está escrito en lenguaje C.

3.2. Ensamblador

No fue difícil encontrar un ensamblador que fuera compatible con la sintaxis del compilador que usábamos en DOS/Windows (MPASM). Usamos el *gpasm*[2] que ya tenía soporte para el 16C716 debido a que el mismo no agrega nuevas instrucciones y lo que el ensamblador agregó fue sólo un archivo de cabecera (header).

3.3. Simulador

Con estas herramientas era posible desarrollar la aplicación propuesta mediante métodos tradicionales, pero siempre es aconsejable probar el desempeño de nuestro programa en un simulador de tiempo real, herramienta disponible en paquetes comerciales importantes.

Si bien esta es la herramienta más compleja se encuentra disponible un simulador llamado *gpsim*. [3] El mismo no sólo simula correctamente el comportamiento de las MCU de Microchip sino que soporta correctamente los dispositivos que incluyen las mismas. Adicionalmente soporta el agregado de *plug-ins* para simular dispositivos externos, entre ellos *displays* LCD y 7 segmentos. La velocidad de simulación es más de un orden de magnitud superior al simulador provisto por el fabricante convirtiéndolo en una excelente herramienta.

El simulador no soportaba los nuevos 16C712/6 así que tuvimos que agregarlo. Gpsim está escrito en C++ y utiliza una estructura de clases muy interesante para describir cada procesador. Se creó una clase derivada de otro controlador muy parecido que agrega las características más importantes de la nueva MCU.

También fue necesario hacer algunos agregados al simulador de LCD para soportar los LCDs seleccionados para esta aplicación.

Si bien fue necesario invertir algo de tiempo el resultado fue más que excelente, en las estaciones de trabajo usadas (K7 750 MHz). La MCU (que en la práctica usa un cristal 4,19 MHz) es simulada a una velocidad superior al tiempo real. Para verificar los tiempo de ejecución y demoras se incorporaron al simulador los comandos e interfaz de usuarios necesarios para medir tiempos.

4. LENGUAJE DE PROGRAMACIÓN

Las MCU de Microchip son muy pequeñas y el desarrollo en lenguajes de alto nivel se complica muchísimo cuando necesitamos explotarlas al máximo. Por otro lado la programación en assembler hace los programas muy difíciles de modificar y por cuestiones de interfaz de usuario y caracterización del sensor a usar era necesario que el programa fuera flexible. Para solucionar esto se optó por usar generadores de código para las partes del programa que debían ser flexibles y assembler para las rutinas de interrupción y apoyo.

Se usaron herramientas que ya habíamos desarrollado con anterioridad y herramientas especialmente desarrolladas para este caso. A continuación se da una breve descripción.

4.1 Statem: generador de máquinas de estado

Statem recibe como entrada un archivo de texto que describe los estados, acciones asociadas y condiciones de transición de una máquina de estados. El lenguaje soporta "supercondiciones" de una manera similar al Graphset de los PLC y operaciones varias para describir las acciones asociadas a cada estado. El mismo puede generar código en lenguaje assembler o C.

4.2 Mifit: Generador de código para aproximar un juego de datos

Mifit[4] permite generar código para aproximar un juego de datos utilizando ecuaciones simples (rectas y parábolas). El mismo permite evaluar el error introducido en la aproximación. Actualmente los datos de entrada son de 8 bits y los cálculos están limitados a 24 bits de resolución.

5. CONCLUSIONES

El uso de herramientas *open source* permite una gran flexibilidad gracias a la disponibilidad del código fuente. Esto nos permitió agregar soporte para una MCU nueva en el mercado, ajustar el comportamiento de las herramientas

a nuestras necesidades (medición de tiempos, display usado, etc.) y corregir problemas en las mismas. Adicionalmente son de extrema utilidad para el caso en cual sea necesario reducir costos, muy común en ambientes educativos de nuestra región que cuentan con recursos muy limitados. La calidad de las herramientas es muy buena y no tienen nada que envidiarle a equivalentes comerciales. Como contrapartida es necesario invertir tiempo para ajustarlas y adaptarlas. En cada situación se deberá evaluar si esto es o no un problema.

En cuanto al uso de herramientas de alto nivel en MCUs tan pequeñas sabíamos que no era recomendable, pero pudimos verificar que generadores de código creados para resolver puntos particulares son de gran ayuda y permiten realizar cambios importantes con mayor facilidad que usando herramientas tales como un compilador de C. El código generado puede optimizarse hasta ser comparable o mejor que el escrito por un humano, con la ventaja de que esto es realizado en una fracción de segundo.

- [1] F. Damgaard, W. Lewis y otros.
<http://home3.inet.tele.dk/frda/picasm/prog.html>
- [2] J. Bowman, S. Dattalo, C. Franklin, J. Cameron y otros.
<http://gpasm.sourceforge.net/>
- [3] S. Dattalo, R. Forsberg, D. Schudel y otros.
<http://www.dattalo.com/gnupic/gpsim.html>
- [4] Salvador E. Tropea, <http://mifit.utic.com.ar/>